FlexDice: 高次元データ空間を持つ大規模データベースに対するリアルタイムクラスタリング

中村 朋健† 上土井陽子†† 若林 真一†† 吉田 典可††

† 広島市立大学大学院 情報科学研究科 〒731–3194 広島市安佐南区大塚東三丁目 4–1
†† 広島市立大学 情報科学部 〒731–3194 広島市安佐南区大塚東三丁目 4–1
E-mail: †tomotake@lcl.ce.hiroshima-cu.ac.jp, ††{yoko,wakaba}@ce.hiroshima-cu.ac.jp

あらまし 高次元かつ大規模なデータベースから規則性や特徴を見つけ出す技法の１つに，類似したデータ要素を集めるクラスタリングがある．本稿では，高次元データ空間上の疎な領域によって分けられる密な領域のデータ要素をリアルタイムに集めるクラスタリング手法 FlexDice を提案する． FlexDice の特徴は２つある． １つ目は，セル構築によって費やされるメモリ領域を極力抑えることである．データ要素の存在しない空間にはセル（格子状に分けられる空間）を構築しない．データ空間上にデータ要素が存在する場合であっても，クラスタリングに不要な疎なセルはメモリ領域から動的に解放できる構造となっている．逆に構築したセル内にデータ要素が多く含まれる密なセルに対してもセルを分割しないことで冗長なセル構築を回避している． ２つ目は，新たに定めたセル間における隣接枝の作成条件により，記憶領域を必要とするセルの隣接枝数が少ないことである．FlexDice は格子構造を用いることで高速に処理でき，セル分割を動的に行うことで冗長なセル構築を回避し，不要になったセルが使用していたメモリを効率よく解放できる．このため従来手法では困難であった 50 次元， 10 万データ要素のような高次元かつ大規模な入力に対して， FlexDice ではリアルタイムな応答が可能である．
キーワード データマイニング，クラスタリング，多次元データ空間，セル，階層構造

FlexDice: Real-Time Clustering for Large Database on High-Dimensional Data Space

Tomotake NAKAMURA†, Yoko KAMIDOI††, Shin'ichi WAKABAYASHI††, and Noriyoshi YOSHIDA††

† Graduate School of Information Sciences, Hiroshima City University
3–4–1, Ozuka-higashi, Asaminami-ku, Hiroshima, 731–3194 Japan
†† Faculty of Information Sciences, Hiroshima City University
3–4–1, Ozuka-higashi, Asaminami-ku, Hiroshima, 731–3194 Japan
E-mail: †tomotake@lcl.ce.hiroshima-cu.ac.jp, ††{yoko,wakaba}@ce.hiroshima-cu.ac.jp

Abstract Clustering is a method of grouping similar objects to find rules and properties from a large high-dimensional database. In this paper, we propose a clustering algorithm *FlexDice*, which groups objects in dense regions separated by sparse ones in multi-dimensional data space. FlexDice has two properties. First, this method needs as few memory used for constructing cells (a part of space separated as a grid) as possible. Even if there are some objects in data space, memories used to store sparse cells can be dynamically freed from memory. Second, FlexDice has only $2 * d$ neighboring edges of one cell by introducing new conditions, where d is the number of dimensions. FlexDice can utilize the advantage of grid-based approaches as reasonably fast and avoid constructing redundant cells by dynamically separating cells. Therefore, for large and high-dimensional input data, such as 50-dimensions and 100,000-objects, FlexDice can realize a real-time response.
Key words Data Mining, Clustering, High-Dimensional Data Space, Cell, Hierarchical Structure

1. はじめに

近年，情報化社会となり，また記憶装置の低価格化が進むことによりデータベースに蓄積されるデータは大規模かつ高次元なものとなっている．今後はさらに大規模かつ高次元なデータベースが構築されることが予想される．大規模かつ高次元なデータベースからデータベースの特徴や規則性を見付け出すことは容易ではない．データベースの特徴や規則性を見付け出すデータマイニングの１つの技法としてクラスタリングがある．クラスタリングはデータベース内における類似したデータ要素を同じクラスタとして集め，類似していないデータ要素を別のクラスタに分けるプロセスのことである．一般的なクラスタリング手法におけるデータ要素の類似性はユークリッド距離やマンハッタン距離のような距離関数によって定められる．また，δ-pCluster [15] ではデータ要素のパターンによって類似性を定めている．

既存のクラスタリング手法は現在の一般的な計算機環境で，50 次元かつ 10 万データ要素をもつような，高次元である大規模な入力データに対して 1 分以内でクラスタリングすることが困難である．本稿では，そのような高次元かつ大規模な入力データに対してリアルタイムにクラスタリングできる手法を開発することを目的とする．ここで本稿におけるリアルタイムとは人が機械に命令を出してその場で何もせずに待てるだろう時間である 30 秒以内に処理を終了することと定義する．

本稿において提案するクラスタリング手法は格子構造を用いる手法であり，その構造によって区切られた空間（セル）が不均一なサイズを持ち，さらに除去も可能であることから FlexDice と呼ぶ．また，FlexDice は密度情報を用いる手法でもあり，d 次元データ空間上における密度の低い疎な領域によって分けられる密度の高い密な領域のデータ要素を集めるクラスタリング手法である．ここで密度は単位体積あたりのデータ要素数とする．

従来の格子構造を用いる手法には STING [16], WaveCluster [14] などがあった．これらの手法は格子構造を用いることで高速に処理できるが高次元データに適用できない．なぜなら，１つ目の理由にこれらの格子構造を用いる手法は入力データの次元数が増加すると１つのセルに対する隣接するセル数が次元数に関して指数関数的に増加するためである [7]．FlexDice はこの問題を，d 次元のセルにおいて，$(d-1)$次元の部分空間を共有するセル同士にのみ隣接関係を作成することで，次元数に指数関数的に増える隣接したセル数を線形関係に抑えて，解決した．ここで，d は入力データの次元数である．2 次元入力データの場合，セルに隣接したセルは８つ存在するが，斜めに点で接するセルには隣接関係を作成せずに辺で接するセルにのみ隣接関係を作成してセルに隣接するセルを４つにとどめる．つまり，１つのセルにおける隣接するセルを d 次元の入力に対して $2d$ 個に抑えた．d 次元のセルにおいて，$(d-1)$ 次元の部分空間を共有するセル同士にしか隣接関係を作成しないことで，クラスタリング精度に多少影響があることが考えられる．しかし，高次元の入力に対して効率的に動くアルゴリズムとなる．

既存の手法が高次元データに適用できないもう１つの理由として，１度に構築したセルへの番号付けができなくなる問題がある．格子構造を用いる手法では，セルを構築した後にクラスタを形成するために隣接する密なセルを結合する必要がある．隣接関係の情報を付加するために，セル構築と同時にセルに番号を付けなければならない．ボトムアップにセルを構築する場合，最初に最も細かいセルを作らなければならず，１つの次元に対して 100 分割することを考えると 50 次元の入力データに対しては 100^{50} 通りのセル番号を用意する必要があり，現実的ではない．空間の各次元に関する 2 等分分割を基本としてセルを構築することで，50 次元であっても 2^{50} の番号を用意することで対応可能となる．

従来の格子構造を用いる手法は１つのデータ要素に対する走査回数を高々 1 回にするため，最初にセルを最も細かく分割していた．その後，ボトムアップにセル構築を進める手法が STING である．一方で，FlexDice は各次元に対して 2 分割することを基本にセル構築を進めていく手法である．再帰的にセル構築を進めることで，１つのデータ要素に対する走査回数は 1 回以上となる．走査回数が 1 回以上になることにより，高速に処理することができなくなる可能性がある．しかし，データ要素が多く含まれた密なセルに対して分割を進めないことで多くのデータ要素に対して再走査せずに済み，高速に処理可能な手法となる．FlexDice は次元に対する 2 等分分割を基本に動的にセルを構築することで，1 度に構築するセル数は高々 2^d 個となり，高次元の入力に対しても効率よくクラスタリング可能となる．

本稿は以下に示す節で構成される．第 2.節において，提案クラスタリング手法 FlexDice に関連した研究を紹介する．第 3.節において，提案手法 FlexDice のアルゴリズムを説明する．第 4.節において，シミュレーション実験に用いる入力データや評価方法について記す．第 5.節において，合成データとベンチマークデータを用いて FlexDice を実験的に評価する．

2. 関連研究

本節以降ではデータ空間は d 次元線形空間と仮定する．クラスタリング手法は文献 [6] により，大きく分類すると分割手法 (*Partitioning methods*)，階層構造を用いる手法 (*Hierarchical methods*)，密度情報を用いる手法 (*Density-based methods*)，格子構造を用いる手法 (*Grid-based methods*)，そしてモデルに基づく手法 (*Model-based methods*) の５つに分類されている．これらの５つに分類される手法の中で

特に FlexDice に関連した密度情報を用いる手法，階層構造を用いる手法，そして格子構造を用いる手法のそれぞれの特徴について第 2.1節で記し，第 2.2節で近年の高次元かつ大規模な入力に対するクラスタリングの研究動向について記す．

2.1 従来手法

1つ目に，任意の形のクラスタを発見するために開発された密度情報を用いる手法がある．密度情報を用いる手法に分類される手法は DBSCAN(Density-Based Spatial Clustering of Application with Noise) [4], OPTICS(Ordering Points To Identify the Clustering Structure) [2]，そして DENCLUE(DENsity-based CLUstEring) [8] などがある．一般的に，これらの手法はデータ空間における低密度の領域によって分けられる高密度の領域のデータ要素をクラスタリングすることを目的とする．密度情報を用いる手法の利点は任意の形のクラスタを見付けることやノイズからクラスタを分けることができることなどである．欠点は入力パラメータを定めることが困難であることなどである．

2つ目に，階層構造を用いる手法がある．階層構造を用いる手法として，BIRCH(Balanced Iterative Reducing and Clustering using Hierarchies) [18], CURE(Clustering Using REpresentatives) [5]，そして Chameleon [9] などがある．これらの手法は結合，または分割のどちらを基本とするかにより，さらに分類される．結合を基本とする手法（agglomerative）は，各データ要素を別々のクラスタとした状態から始め，図 1の上部の矢印で示すように結合を進める．データ要素のすべてが階層構造の最上位レベルのクラスタに併合されるか，与えられた終了条件を満たすまで，データ要素，またはクラスタは他の近いものから連続的に併合される．図 1の下部の矢印で示すような分割を基本とする手法（divisive）は，反復によって最終的に各データ要素が1つのクラスタになるか，与えられた終了条件を満たすまで連続的に分割される．階層構造を用いる手法の利点は予めクラスタ数を決める必要がないこと，利用者の要求に合わせて複数の結果を出力できること，そして一般的に精度が高いことなどである．欠点は計算時間が遅いことである．一般的に階層構造を用いる手法の計算複雑さは $O(n^2 \log n)$ である [13].

3つ目に，高速に処理できる格子構造を用いるクラスタリング手法がある．格子構造を用いる手法として，STING(STatistical INformation Grid)， WaveCluster, そして CLIQUE(CLustering In QUEst) [1] などがある．これらの手法は個々のデータ要素を結合させてクラスタを発見するのではなく，多重解像度の格子データ構造を用いてセル単位でクラスタを発見する手法である．セルとはデータ空間の各次元に関し 2 つの値を定めることによって，各

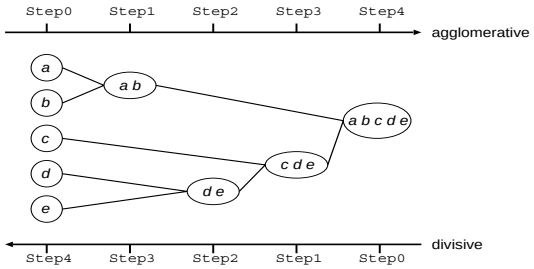

図 1 階層クラスタリングにおけるデータ要素 $\{a, b, c, d, e\}$ の結合と分割

座標軸に平行に分けられるデータ空間のことである．格子構造を用いる手法では各次元における座標を K 個の区間に分割する．d 次元データに対しては，K^d 個のセルに分割し，類似した性質のセルを集めることでクラスタリングする手法である．格子構造を用いる手法の主な利点は計算時間が速いことである．結合のための計算時間がデータ要素数に依存せず，量子化されたセル数のみに依存する．一般的に，データ要素のセルへの割り当てとデータ要素集合によるクラスタ表現を除くと，これらの手法の計算複雑さは $O(g)$ である．ここで g はデータ空間上のセル数である．欠点としてクラスタの縁を曲線に出来ないことやメモリ使用量が多いことなどがある．クラスタの縁とはデータ空間上で生成されるクラスタとノイズの境目である．クラスタの縁を曲線にできない理由として，高速に処理するためにセル単位でデータ要素を扱うことにより，クラスタの縁が角張った形になるためである．メモリ使用量に関しては，他の手法には必要ないセルを格納する領域を必要とするため，他の手法よりも多くなる．特に，高次元データが入力であると通常セル数が莫大に増える可能性があることや，隣接したセルを結合させる際に多くの計算時間を必要とすることがある．

2.2 高次元データを対象とする手法の動向

近年，大規模かつ高次元な入力に対するクラスタリング手法 [7], [15], [11], [10], [17] は数多く発表されている．目的とするクラスタはそれぞれ異なったものが多いが，それぞれのクラスタリング手法は 50 次元かつ 10 万データ要素を越える入力に対してクラスタリングを試みていて，計算時間の高速化を目標としている．しかし，次元数や要素数に線形関係の計算時間であるものはあるが，どの手法も 50 次元かつ 10 万データ要素の入力に対してリアルタイムに処理できていない．

O-Cluster [11] の目的とするクラスタは密な連続した領域のデータ要素を集めることである． O-Cluster は格子構造を用いる手法であり，データ要素の走査は 1 度に抑えて，データ要素が密集していないところを選びデータ空間を分割する．これらの特徴は OptiGrid [7] と同様である．O-Cluster や OptiGrid では高次元な入力において 1 点で

接するセル同士に隣接関係が必要であると考えられている．そこで，次元数に関し指数関数的に増加する隣接するセルを持たないように，分割面を選択してデータ空間を分割する．つまり，OptiGrid や O-Cluster は分割する面を選び分割するため，隣接したセル同士にもあまり強い関係がない．したがって，各セルに対する隣接関係の作成を必要とせず，１つのセルに対して $2d + 2^d$ 個の隣接したセルを持たない．図 2は 2 次元入力データに対する O-Cluster の空間分割後のセル構造の例である．

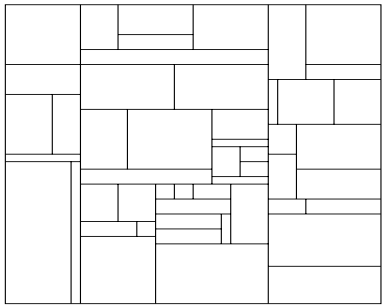

図 2　O-Cluster のセル構造の例（2 次元）

3．FlexDice アルゴリズム

FlexDice は d 次元データ空間上における低密度の領域によって分けられる高密度な領域のデータ要素を高速に集めることを目的とするクラスタリング手法である．高速に処理するために，データ要素一つ一つを扱ってクラスタリングするのではなく，データ空間を平面により分けることで構成されるセルを用いてクラスタリングする．通常，このような手法は格子構造を用いる手法に分類される． FlexDice はセル分割を動的に行う手法である．まず，データ要素が存在する全空間を１つのセルと考える．次に，各次元における空間の中心を軸垂直に 2 分割するすることで新しい子セルを構築する．このとき最大で 2^d 個のセルが構築されるが，セルを構築することで費やされるメモリを削減するために，新たに構築する子セル内にデータ要素があるときのみセルを構築する．データ要素を子セルに振り分けた後に，構築したセルの密度を調べ疎セルであった場合は，そのセルに含まれていたデータ要素をノイズと判断して，メモリを費やしていたセルを解放する．ここで，疎セルとはパラメータによって疎と判断されたセルのことである．また密セルとはパラメータによって密と判断されたセルであり，中セルとは密セルでも疎セルでもないセルのことである．疎セルの解放もメモリ使用量を削減するためである．特に入力データが高次元データである場合，格子構造を用いる手法はメモリ使用量について細心の注意を払わなければならない．

FlexDice ではメモリ使用量を削減するもう１つの方策として，密セルまたは疎セルには子セルを構築しない．密セルまたは疎セルに対して子セルを構築しないことにより最終的にできあがったセルは不均一なサイズのセルとなる．セルのサイズを不均一にすることにより，１つのセルに対して隣接するセル数を次元数に依存する数以内に限定できなくなる．図 3は 2 次元入力データに対するセル構築後のセル構造である．最も小さいセルに対して，隣接したセルは４つ存在するが，最も大きいセルに対しては８つとなっている．図 3は 2 次元の入力であり 4 階層までの分割としたため差は少ないが，d 次元の入力であり l 階層までの分割とすると，隣接したセルは最小で $2d$ 個であり最大で $2^l d$ 個となる．隣接関係の作成はセルに保持する隣接関係の枝数を最大数に合わせて $2^l d$ 本用意することで可能となるが，高次元かつ高階層で実行する場合，隣接枝数が多くなるため効率的でない．そこで，隣接するセルが自分以上のサイズのセルである場合にのみ隣接関係を作成することで，各セルの隣接枝数を階層数には関係なく $2d$ 本以下に限定した．

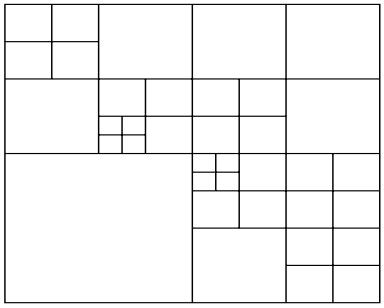

図 3　FlexDice におけるセル構築後のセル構造の例（2 次元）

さらに，各階層毎に隣接関係を作成することにより，子セルを作成した中セルのメモリ領域は解放可能となる．また，各階層毎に隣接関係を作成することにより，新たに隣接関係を作成する際に辿る枝は 3 本以下となる．隣接関係を作成するときの辿る枝のパターンを以下に示す．

- 対象のセル → 親セル → 隣接するセル
- 対象のセル → 親セル → 親セルに隣接するセル
  → 隣接するセル

同じ親セルを持つ子セル同士の隣接関係を作成する場合と親セルに隣接したセルが密セルまたは疎セルへの隣接関係を作成する場合は，上記の１つ目の方法により 2 本の枝を辿ることで隣接関係を作成できる．親セルに隣接したセルが中セルである場合は，上記の２つ目の方法により 3 本の枝を辿ることで隣接関係を作成できる．隣接関係を作成するときに辿る枝数を 3 本以下と限定することで，高速に隣接関係を作成可能となる．

FlexDice は第 2.1節において紹介した各手法の良い特徴を含んでいる．つまり，FlexDice は格子構造を用いる手法の高速性を維持したまま，メモリの使用量を削減して階層的手法の汎用性を活かし，低密度の領域によって分けられ

る連続した高密度のデータ要素をクラスタとして集める手法である．

FlexDice は，以下の２つのフェーズから構成される．フェーズ１ではデータ空間を分割しながら各階層毎に隣接関係を作成し，フェーズ２では隣接した密セルを結合することでクラスタを形成する．

3.1 フェーズ１ — 分割と接続 —

入力パラメータ $PMin$，$PMed$，$PMax$，$MinC$ を用いた FlexDice アルゴリズムのフェーズ１を図 4に示す．セルの密度が $PMin$ より小さければ疎セル，$PMax$ 以上なら密セルとし，$PMin$ 以上 $PMax$ より小さいセルを中セルとする．セルの辺の長さの少なくとも１つが $MinC$ より小さいセルに対しては，密度が $PMed$ 以上のセルを密セル，$PMed$ より小さければ疎セルと呼ぶ．

(1) 入力データ全体の密度情報を調べる．$PMax$ 以上であれば，入力データ全体を一つのクラスタとし終了する．$PMin$ より小さければ，入力データにクラスタは存在しないとし終了する．$PMin$ 以上かつ，$PMax$ より小さければ，そのセルをキューに入れ (2) へ進む．

(2) キューにセルが含まれていれば (3) へ進む．含まれていなければ終了する．

(3) 取り出したセルの大きさが $MinC$ 以上であれば (4) へ進む．$MinC$ より小さければ (5) へ進む．

(4) 取り出したセルの密度情報を調べる．$PMax$ 以上であれば，密セルと判断し (2) へ進む．$PMin$ より小さければ，疎セルと判断し (2) へ進む．$PMin$ 以上かつ，$PMax$ より小さければ，セルを 2^d 分割し，隣接関係をグラフ化する．分割されたセルをキューに入れ (2) へ進む．

(5) キューに含まれる各セルに対し密度が $PMed$ 以上であれば，密セルと判断し，$PMed$ より小さければ疎セルと判断する．

図 4　FlexDice アルゴリズム（フェーズ１）

セルの分割は中セルに対してのみ行う．各次元に対して 2 分割するため，d 次元データ空間上のセルに対しては各セルを 2^d 分割することになる．しかし，高次元データの場合，多くの分割された空間でデータ要素が存在せずセルを構築しない．また，多くのセルが疎セルとなりそれらのメモリ領域を解放できる．構築された子セルは親から高速に辿れるようにするために，ハッシュ関数を用いて格納される．図 5に，２次元データに対する FlexDice の階層構造を示す．

以下で図 5の例を用い，FlexDice のフェーズ１の処理を説明する．図 5は第１層（1st layer）から始まり，第 i 層で終わる．第１層では疎密の判断ができないので４分割する．第１層に散らばったデータ要素を走査し，分割された第 2 層のセルを構築しながらデータ要素を振り分ける．しかし，第 2 層における左奥のセルに振り分けられるデータ要素は存在しないため，セル構築はしない．第 2 層におけるセルが構築された状態で，構築したセルから隣接するセルへの単方向の枝の情報をセルに保持させる．第 2 層の右手前のセルに関する隣接セルは３分割された残りの２つのセルと，１つ上の第１層に戻り第１層の親セルに隣接した第１層のセル，または第１層のセルに隣接したセルの第 2 層のセルを辿ることで隣接関係を作成できる．第 2 層の隣接関係作成後には第１層のセルのために確保したメモリを解放する．第 2 層では，密セル，または疎セルに対しては，これ以上階層数を増加させずに，中セルに対してのみ分割する．同様の操作を繰り返し第 i 層まで行う．

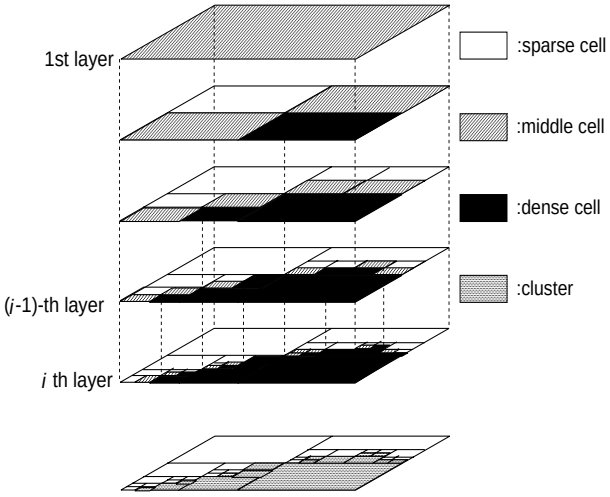

図 5　FlexDice の階層構造（2 次元）

以上のように隣接関係を作成すると，隣接関係は同一層，またはそれより階層数の少ない密セルに対してのみの関係となる．つまり，セルは自分のサイズ以上の隣接した密セルにのみ隣接関係を作成する．２次元データにおける作成された隣接関係を図 6に示す．ここで，網がかったセルが密セル，矢印がセル間の隣接関係を表す．

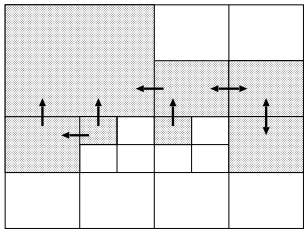

図 6　作成された隣接関係の例（2 次元データ）

3.2 フェーズ２ — 結合 —

フェーズ２では構築したセルと作成した隣接関係から，隣接する密セルを結合したセル集合としてクラスタを形成する．フェーズ２の手順は，まず，任意の密セルから辿れるすべての密セルを集めてサブクラスタを形成する．次に，サブクラスタ間の隣接関係を双方向な接続関係に拡張し，最終的なクラスタを形成する．FlexDice アルゴリズムのフェーズ２を図 7に示す．

(1) サブクラスタに属していないセルを任意に選び (2) へ進む．存在しなければ (3) へ進む．
(2) 作成した隣接関係から辿れるすべての密セルを集めサブクラスタとする．セルからサブクラスタに辿る場合は接続関係を保持して，サブクラスタから先の隣接関係を探索しない．(1) へ戻る．
(3) (2) で保持させたサブクラスタ間の接続関係を双方向にする．(4) へ進む．
(4) クラスタに属していないサブクラスタを任意に選び，(5) へ進む．存在しない場合，終了する．
(5) 辿れるサブクラスタを結合しクラスタを形成する．(4) へ戻る．

図 7　FlexDice アルゴリズム（フェーズ 2）

4. 実験の準備

第 5.節で FlexDice を実験的に評価するためのシミュレーション実験を行う前に，入力に用いるデータや手法の評価方法について説明する．

4.1 入力データの種類

シミュレーション実験に使用する入力データは合成データと実データに基づいて作成されたベンチマークデータを用いる．合成データは計算機上で乱数を発生させて作成している．合成データにおけるクラスタは発生する乱数の値域を狭めて多くの乱数を発生させることで作成したものであり，幅広い値域で発生させたノイズのデータと結合させることで入力データを作成した．ベンチマークデータには保険会社の実際の顧客データに基づくベンチマークデータ（The Insurance Company Benchmark）[19] を用いる．保険会社のベンチマークデータは入力データ要素数 4,000，次元数 85 をもつ．

4.2 クラスタリング結果の評価方法

クラスタリング結果を評価するための誤差と計算時間を定義する．

まず，誤差を定義する．通常，クラスタリングに用いる入力データはデータの分析ができていないため，目的関数を明確に示すことは非常に難しい問題である．しかし，今回使用する 2 次元の合成データはクラスタとノイズを予め決めて作成したものであるため，誤差の評価が可能である．

クラスタリング結果を評価するために誤差 ERR は

$$ERR = \frac{I_{num} - T_{num}}{I_{num}}$$

とする．ここで I_{num} は入力データの全データ要素数であり，T_{num} は正しくクラスタリングされた全データ要素数である．誤差の値域は 0 以上，1 以下であり，0 はクラスタリングの結果が最も良質なときの値である．複数のクラスタを 1 つのクラスタと認識した場合や，1 つのクラスタを複数のクラスタと認識した場合は正しくクラスタリングされた要素数を数えることができない．そこで，入力データのクラスタの集合 V_1，出力結果のクラスタの集合 V_2，入力データのクラスタの集合と出力結果のクラスタの集合に対応するデータ要素数を重みとする枝の集合 E で構成されるグラフ $G = (V_1, V_2, E)$ の最大マッチングを利用して正しくクラスタリングされた要素数を数える．

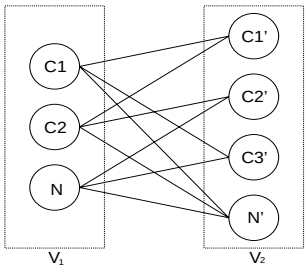

図 8　誤差を求めるためのグラフの例

図 8はグラフ $G = (V_1, V_2, E)$ の例であり，このグラフの利益が最大のマッチングにおける利益を，正しくクラスタリングされた要素数とする．ここで利益とは枝の重みであり，この場合入力データから出力結果へ遷移したデータ要素数である．

次に，計算時間を定義する．計算時間は入力データを主記憶上に取り込んだ後から主記憶上にクラスタが形成されるまでの時間とする．つまり，フェーズ 1 とフェーズ 2 に費やした CPU 時間である．第 5.節における図では計算時間を Time と表記する．

5. シミュレーション実験と考察

FlexDice と STING を C 言語を用いて SUN Ultra60 Model1450（CPU:450MHz，主記憶装置:1GB）上に実現し，シミュレーション実験を行った．

5.1 実験 1 —FlexDice と STING の比較—

入力データとして 2 次元描画データを用いた．入力データには，あらかじめクラスタを形成したデータ要素グループと多くのノイズが含まれている．出力結果において，同じクラスタに属するデータ要素は同パターンで表し，ノイズは出力しないものとする．

図 9の入力データを用いて，FlexDice と STING の計算時間と構築セル数について比較する．ここで，両手法の誤差 ERR が 0.001 以下であるときの比較とする．図 9では見えにくいが，入力データ 1 には多くのノイズを含んでいる．

FlexDice と STING の出力結果をそれぞれ図 10，図 11に示し，これらを出力したときの両手法の計算時間と構築セル数を表 1に示す．

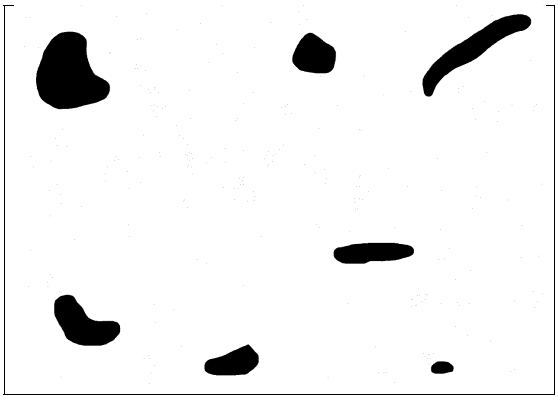

図 9　入力データ 1（要素数:489,386）

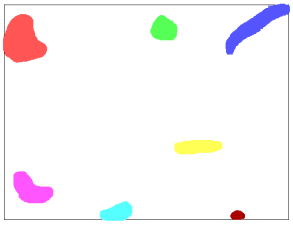

図 10　FlexDice の出力結果

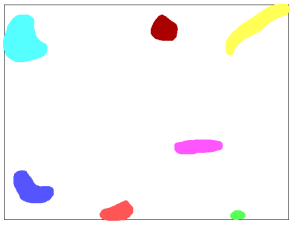

図 11　STING の出力結果

表 1　FlexDice と STING の計算時間と構築セル数

	計算時間 [s]	構築セル数
FlexDice	2.98	4,630
STING	9.28	349,525

両手法とも，最下位層でのセルを小さくすることで高精度な結果が得られた．そのときの計算時間を比較すると FlexDice は STING よりも 3 倍以上高速であった．これは FlexDice が冗長なセルを構築しなかったことにより現れた差と考えられる． FlexDice のフェーズ 1 に費やした計算時間は 2.978[s] であり，フェーズ 2 は 0.006[s] であった．

構築セル数において， 2 次元データで 10 階層まで下げた場合， STING は FlexDice の約 80 倍ものセルを必要としているため，メモリ使用量に大きな差が出る．高次元な入力データであると，さらに大きな差がでることが容易に分かる．メモリ使用量を大きく左右する構築セル数を減らせたことで， FlexDice はさらに大規模な入力データに対しても高速，かつ高精度な結果を出力できる．

5.2　実験 2 — データ要素数と計算時間の関係 —

データ要素数と計算時間の関係を調べるために，次元数 10，データ要素数 10 万から 100 万までの合成データを入力として，データ要素数と計算時間の関係を調べる．出力はどのデータにおいても作成したクラスタを認識したときのものとしている．データ要素数と計算時間の関係を図 12に示す．

図 12の実験結果から，入力とした合成データに関してデータ要素数と計算時間は比例関係にあることが分かる．

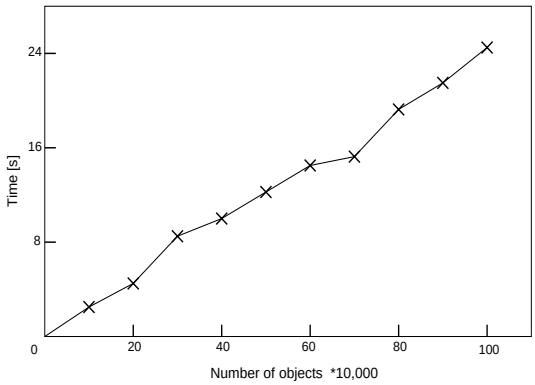

図 12　10 次元の入力データに関するデータ要素数と計算時間の関係

また， 10 次元の入力データに関してはデータ要素数 100 万まではリアルタイムに処理できることが分かる．

5.3　実験 3 — 実データへの適用 —

保険会社の顧客情報データをクラスタリングすることで有益な情報を得られることが多い．例えば，契約してくれる（または契約してくれない）顧客のグループを発見することだけでなく，事故を起こしやすい（または起こしにくい）グループを発見することや保険料の効率的な値段設定など様々な有益な情報を得ることが可能となる [3]．このようなデータをリアルタイムに処理できるようにすることで，動的に変化するデータベースを保険料金の表にすることなく，その場で会社にとっても顧客にとっても満足する料金を弾き出せるようになる．

保険会社のベンチマークデータは 85 次元であるが，結果的に 2 値データとなる項目が約 30 項目ある． FlexDice は 2 値データを入力データの一部とすると， 2 階層までしかセルを分割することができない．したがって，本実験においては 2 値データを除外した 10 次元から 50 次元までの実験結果を図 13に示す．どの出力結果も様々な大きさで，いくつかの意味のありそうなクラスタが形成されたときのものとなっている．

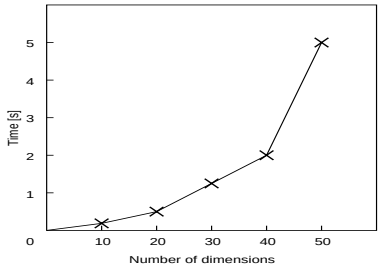

図 13　ベンチマークデータに関する次元数と計算時間の関係

例えば， 50 次元の入力に対する出力結果では 30 データ要素を越える 8 つのクラスタを見つけ出している．これらのクラスタは類似した値のデータ要素で構成されており， 2 次元の出力結果のように意味のありそうなクラスタが形成

されている．図13の結果から，FlexDice は 50 次元の高次元データに適用可能であることが分かる．また，計算時間は 50 次元であっても約 5 秒であるため，すべてリアルタイムに処理できていることが分かる．

6. おわりに

本稿では高次元かつ大規模データベースに対してリアルタイムにクラスタリングすることを目的とした FlexDice を提案した．シミュレーション実験では同じ手法に分類されるだろう FlexDice と STING の性能について比較，FlexDice におけるデータ要素数と計算時間の関係，そして実データに基づくベンチマークデータにおける FlexDice の次元数と計算時間の関係を調べた．

STING は高次元入力データに不向きなことは容易に予測できる．2 次元の入力データを用いて計算時間とメモリ使用量について比較したが，低次元の入力データにおいても計算時間やメモリ使用量に関して STING よりも FlexDice が効率的であることを示した．

FlexDice は 10 次元であり大規模な入力データに対して，計算時間とデータ要素数の関係を線形に抑えることができ，10 次元の入力に対して 100 万データ要素までリアルタイムに処理できることを示した．

実データを用いたシミュレーション実験では入力データ要素数は少ないものの，有益な情報を多く含んでいそうな結果をリアルタイムに出力できた．

今後の課題として，d 次元のセルにおいて，$(d-1)$次元の部分空間を共有するセル同士にしか隣接関係を作成しないことで生じた精度の違いを検証すること，今回のベンチマークデータではシミュレーションできなかった高次元である大規模な実データに対して実験すること，クラスタリング結果の有効性を示すこと，パラメータと出力の関係を検証することなどがある．

文　献

[1] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan: “Automatic subspace clustering of high dimensional data for data mining applications,” Proc. 1998 ACM-SIGMOD Int. Conf. on Management of Data (SIGMOD’98), pp-428-439, 1998.

[2] M. Ankerst, M. M. Breunig, H. -P. Kriegel, and J. Sander: “OPTICS:Ordering points to identify the clustering structure,” Proc.1999 ACM-SIGMOD Int. Conf. on Management of Data (SIGMOD’99), pp.49-60, 1999.

[3] C. Apte, E. Grossman, E. Pednault, B. Rosen, F. Tipu, and B. White: “Probabilistic estimation based data mining for discovering insurance risks,” IEEE Intelli. Syst. 14, 6, pp.49-58, 1999.

[4] M. Ester, H. -P. Kriegel, J. Sander and X. Xu: “A density-based algorithm for discovering clusters in large spatial databases with noise,” Proc.1996 Int. Conf. Knowledg Discovery and Data Mining (KDD’96), pp.226-231, 1996.

[5] S. Guha, R. Rastogi, and K. Shim: “CURE: An efficient clustering algorithm for large database,” Proc. 1998 ACM-SIGMOD Int. Conf. on Management of Data (SIGMOD’98), pp.73-84, 1998.

[6] J. Han and M. Kamber: “Data Mining: Concepts and Techniques,” Academic Press, pp. 335-376, 2001.

[7] A. Hinneburg and D. A. Keim: “Optimal grid-clustering: Towards breaking the curse of dimensionality in high-dimensional clustering,” Proc.25th Int. Conf. on Very Large Data Bases (VLDB’99), pp.506-517, 1999.

[8] A. Hinneburg and D. A. Keim: “An efficient approach to clustering in large multimedia databases with noise,” Proc. 1998 Int. Conf. Knowledge Discovery and Data Mining (KDD’98), pp.58-65, 1998.

[9] G. Karypis, E. -H. (Sam) Han, and V. Kumar: “CHAMELEON: A hierarchical clustering algorithm using dynamic modeling,” IEEE Computer, 32, 8, pp.68-75, 1999.

[10] J. Liu and W. Wang: “OP-Cluster: Clustering by tendency in high dimensional space,” Proc. 2003 IEEE Int. Conf. on Data Mining (ICDM’03), pp.187-194, 2003.

[11] B. L. Milenova and M. M. Campos: “O-Cluter: Scalable clutering of large high dimensional data sets,” Proc. 2002 IEEE Int. Conf. on Data Mining (ICDM’02), pp.290-297, 2002.

[12] J. Pei, X. Zhang, M. Cho, H. Wang, and P. S. Yu: “MaPle: A fast algorithm for maximal pattern-based clustering,” Proc. 2003 IEEE Int. Conf. on Data Mining (ICDM’03), pp.259-266, 2003.

[13] J. Sander: “Principles of Knowledge Discovery in Data,” http://www.cs.ualberta.ca/ joerg/courses /cmput695/fall2003/.

[14] G. Sheikholeslami, S. Chatterjee, and A. Zhang: “WaveCluster: A multi-resolution clustering approach for very large spatial databases,” Proc. 24th Int. Conf. on Very Large Data Bases (VLDB’98), pp.289-304, 1998.

[15] H. Wang, W. Wang, J. Yang, and P. S. Yu: “Clustering by pattern similarity in large data sets,” Proc. 2002 ACM-SIGMOD Int. Conf. on Management of Data (SIGMOD’02), pp.394-405, 2002.

[16] W. Wang, J. Yang, and R. Muntz: “STING:A statistical information grid approach to spatial data mining,” Proc. 1997 Int. Conf. Very Large Data Bases (VLDB’97), pp.186-195,1997

[17] J. Yang, W. Wang, H. Wang, and P. Yu: “δ-Cluster: Capturing subspace correlation in a large data set,” Proc. 2002 Int. Conf. on Data Engineering (ICDE’02), pp.517-528, 2002.

[18] T. Zhang, R. Ramakrishnan, and M. Livny: “BIRCH: An efficient data clustering method for very large,” Proc. 1996 ACM-SIGMOD Int. Conf. on Management of Data (SIGMOD’96), pp.103-114, 1996.

[19] The University of California, Irvine Knowledge Discovery in Databases Archive: “The insurance company benchmark (COIL 2000),” http://kdd.ics.uci.edu/.